ICS 311: Algorithms Spring 2025

Problem Set 9

Prof. Nodari Sitchinava *Due: Friday, Mar 14, 2025 at 4pm*

You may discuss the problems with your classmates, however **you must write up the solutions on your own** and **list the names** of every person with whom you discussed each problem.

Start **every** problem on a separate sheet of paper, with the exception of Problems 1 (Peer credit assignment) – Problem 1 can be on the same page as any other problem. Any problem that starts on the same sheet of paper as some other problem will receive 0 points!

1 Peer Credit Assignment (1 point extra credit for replying)

Please list the names of the other members of your peer group for this week and the number of extra credit points you think they deserve for their participation in group work.

- You have a total of 60 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- ***You cannot allocate any points to yourself!*** Points allocated to yourself will not be recorded.

2 Moving on a Checkerboard (30 pts)

Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

1. the square immediately above
2. the square that is one up and one to the left (but only if the checker is not already in the leftmost column)
3. the square that is one up and one to the right (but only if the checker is not already in the rightmost column)

Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given a list of the values $p(x, y)$ for each pair (x, y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

(a) **(14 pts)** Design a ***recursive backtracking*** (brute-force) algorithm that determines the maximum amount of money you can receive, when moving a checker from somewhere on the bottom row to somewhere on the top row. Your algorithm is free to pick any square along the bottom row as a starting point and any square along the top row as a destination in order to maximize the number of dollars gathered along the way. Write down the pseudocode of your algorithm.

(b) **(8 pts)** Modify your pseudocode in part (a) to use memoization. Write down the modified pseudocode.

(c) **(8 pts)** Design a polynomial-time bottom-up dynamic programming algorithm for the above problem. Write down the pseudocode and analyze the running time. ***Argue why your choice of the array and the order in which you fill in the values is the correct one.***

3 Coin Changing (40 pts)

Consider the problem of making change for n cents using the fewest number of coins. Assume that we live in a country where coins come in k different denominations $c_1, c_2, \ldots, c_k$, such that the coin values are positive integers, $k \geq 1$, and $c_1 = 1$, i.e., there are pennies, so there is a solution for every value of n. For example, in case of the US coins, $k = 4, c_1 = 1, c_2 = 5, c_3 = 10, c_4 = 25$, i.e., there are pennies, nickels, dimes, and quarters. To give optimal change in the US for n cents, it is sufficient to pick as many quarters as possible, then as many dimes as possible, then as many nickels as possible, and finally give the rest in pennies.

(a) **(2 pts)** Give an example of coin denominations, such that using the above strategy of picking the maximum number of coins of the largest denomination will not lead to an optimal solution, i.e. will not minimize the number of coins in the change.

(b) **(8 pts)** Prove that the coin changing problem exhibits optimal substructure.

(c) **(10 pts)** Design a ***recursive backtracking*** (brute-force) algorithm that returns the ***minimum number of coins*** needed to make change for n cents for ***any*** set of k different coin denominations. Write down the pseudocode and prove that your algorithm is correct. ***No points will be given for an algorithm without the proof.***

(d) **(5 pts)** Modify your pseudocode in part (c) to use memoization. Write down the modified pseudocode and ***expalin your choice*** of the table dimensions and the indices used for memoization. ***A solution without the explanation will receive 0 points!***

(e) **(5 pts)** Design a bottom-up (non-recursive) $O(nk)$-time algorithm that makes change for any set of k different coin denominations. Write down the pseudocode and analyze its running time. ***Argue why your choice of the array and the order in which you fill in the values is the correct one.***

(f) **(5 pts)** Suppose now that the available coins are in the denominations that are powers of c, i.e., the denominations are c^0, c^1, ..., c^k for some integers $c > 1$ and $k \geq 1$.

Write down pseudocode for a ***greedy*** algorithm that returns the fewest number of coins needed for making change for n cents. Analyze the running time of your algorithm.

(g) **(5 pts)** Prove that your greedy algorithm always returns an optimal solution for coin denominations that are powers of c. *Hint: prove the greedy choice property. Don't forget to state explicitly the precise claim you are proving before proving it.*

4 Printing neatly (30 pts)

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the end of the line is $S_{i,j} = M - j + i - \sum_{k=i}^{j} l_k$, which must be nonnegative so that the words fit on the line. We wish to minimize the sum of $(S_{i,j})^3$, over all lines except the last. That is, we want to minimize the sum of the cubes of the numbers of extra space characters at the ends of all lines, except the last one. (You can think of $(S_{i,j})^3$ as the penalty for leaving spaces at the end of the line.)

(a) **(10 pts)** Prove that the problem of printing a paragraph of n words neatly on a printer exhibits optimal substructure.

(b) **(10 pts)** Design a ***recursive backtracking*** algorithm to print a paragraph of n words neatly on a printer. Don't forget the special case of how penalty is calculated for the last line to be printed.

Write down the pseudocode and ***explain why it correcly finds the optimal solution***. A solution with no explanation will receive 0 points! Make your pseudocode ***as simple as possible***. You will be graded on the elegance of your solution, not just the correctness.

(c) **(5 pts)** Modify your pseudocode in part (b) to use memoization. Write down the modified pseudocode and ***explain your choice*** of the table dimensions and the indices used for memoization. ***A solution without the explanation will receive 0 points!***

(d) **(5 pts)** Convert the memoized solution in part (c) into a bottom up (non-recursive) solution. Write down the pseudocode and analyze the running time and space requirements of your algorithm. ***Argue*** why your choice of the order in which you fill in the table values is the correct one.

(e) **(OPTIONAL - 0 pts)** How can your solution be simplified if the penalty for leaving space at the end of the line was simply $S_{i,j}$, instead of $(S_{i,j})^3$.

5 Parenthesization Puzzle (OPTIONAL - 0 pts)

Your local newspaper started publishing puzzles of the following form:

Parenthesize $6 + 0 \cdot 6$ to maximize the outcome.	Parenthesize $0.1 \cdot 0.1 + 0.1$ to maximize the outcome.

Wrong answer: $6 + (0 \cdot 6) = 6 + 0 = 6$.
Right answer: $(6 + 0) \cdot 6 = 6 \cdot 6 = 36$.

Wrong answer: $0.1 \cdot (0.1 + 0.1) = 0.1 \cdot 0.2 = 0.02$.
Right answer: $(0.1 \cdot 0.1) + 0.1 = 0.01 + 0.1 = 0.11$.

To save yourself from tedium, but still impress your friends, you decide to implement an algorithm to solve these puzzles. The input to your algorithm is a sequence

$$x_1, o_1, x_2, o_2, \ldots, x_n, o_n, x_{n+1}$$

consisting of $n + 1$ non-negative real numbers $x_1, x_1, \ldots, x_{n+1}$ and n operators $o_1, o_2, \ldots, o_n$. Each operator o_i is either addition $(+)$ or multiplication $(\cdot)$. (If it helps, you can view the input given to you as two separate arrays $X[1..(n + 1)]$ and $O[1..n]$).

(a) Prove that the above problem exhibits optimal substructure.

(b) Design a ***recursive backtracking*** (brute-force) algorithm that computes the maximum outcome for a given input of numbers and operators. Don't worry about making it fast for this part.

(c) Modify your pseudocode in part (b) to use memoization. Write down the modified pseudocode.

(d) Design a polynomial-time bottom-up dynamic programming algorithm for finding the optimal (maximum-outcome) parenthesization of the given expression, write down its pseudocode and analyze the running time. ***Argue why your choice of the array and the order in which you fill in the values is the correct one.***

(e) What additional information do you need to output actual parenthesization and not just the value of the outcome? Can you modify your algorithm to return the actual parenthesization?

6 Minimum Palindrome Supersequence (OPTIONAL - 0 pts)

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `CIVIC`, `RACECAR`, `NOON`, and `AIBOHPHOBIA` (fear of palindromes).

In this problem, given a string s, you are asked to find the ***length*** of the ***shortest*** string that is a palindrome and contains s as a subsequence. For example, given the string s = `TWENTYONE` you should return 13, because the palindrome `TWENTOYOTNEWT` contains s as a subsequence and consists of 13 characters (and there is no shorter string that satisfies these conditions).

(a) Design a ***recursive backtracking*** algorithm that takes a string as an argument and returns the ***length*** of the shortest palindrome supersequence. Make your algorithm as simple as possible.

You may assume that an input string s is given as an array of characters. For example, input string `NOON` is given as an array $s[1..4]$, where $s[1]$ = `N`, $s[2]$ = `O`, $s[3]$ = `O`, and $s[4]$ = `N`.

Write down the pseudocode and prove that your algorithm is correct. *Hint: use induction.* ***No points will be given for an algorithm without the proof.*** If you are struggling to prove the correctness of your algorithm, chances are it is either too complicated, is incorrect, or both.

(b) Modify your pseudocode in part (a) to use memoization. Write down the modified pseudocode and ***explain your choice*** of the table dimensions and the indices used for memoization. ***A solution without the explanation will receive 0 points!***

(c) Convert the memoized solution in part (b) into a bottom up (non-recursive) solution. Write down the pseudocode and analyze the running time and space requirements of your algorithm. ***Argue*** why your choice of the order in which you fill in the table values is the correct one.